Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 390 (2008) 248–270
Theoretical Computer Science
www.elsevier.com/locate/tcs

# Free-algebra models for the $\pi$-calculus

Ian Stark

*Laboratory for Foundations of Computer Science, School of Informatics, The University of Edinburgh, Scotland, United Kingdom*

**Abstract**

The finite $\pi$-calculus has an explicit set-theoretic functor-category model that is known to be fully abstract for strong late bisimulation congruence. We characterize this as the initial free algebra for an appropriate set of operations and equations in the enriched Lawvere theories of Plotkin and Power. Thus we obtain a novel algebraic description for models of the $\pi$-calculus, and validate an existing construction as the universal such model.

The algebraic operations are intuitive, covering name creation, communication of names over channels, and nondeterministic choice; the equations then combine these features in a modular fashion. We work in an enriched setting, over a "possible worlds" category of sets indexed by available names. This expands significantly on the classical notion of algebraic theories: we can specify operations that act only on fresh names, or have arities that vary as processes evolve.

Based on our algebraic theory of $\pi$ we describe a category of models for the $\pi$-calculus, and show that they all preserve bisimulation congruence. We develop a direct construction of free models in this category; and generalise previous results to prove that all free-algebra models are fully abstract. We show how local modifications to the theory can give alternative models for $\pi\mathrm{I}$ and the early $\pi$-calculus.

From the theory of $\pi$ we also obtain a Moggi-style computational monad, suitable for a programming language semantics of mobile communicating systems. This addresses the challenging area of correctly combining computational monads: in this case those for concurrency, name generation, and communication.


*Keywords:* Denotational semantics; Concurrency; Computational monads; Lawvere theories; Nominal sets

## 1. Introduction

There are by now a handful of models known to give a denotational semantics for the $\pi$-calculus [4,5,8–10,12,45]. All are fully abstract for appropriate operational equivalences, and all use functor categories to handle the central issue of names and name creation. In this article we present a method for generating such models purely from their desired algebraic properties.

We address specifically the finite $\pi$-calculus model as presented by Fiore et al. [10]. This uses the functor category $\mathcal{S}et^{\mathcal{I}}$, with index $\mathcal{I}$ the category of finite name sets and injections, and is fully abstract for strong late bisimulation congruence. We exhibit this as one among a category of algebraic models for the $\pi$-calculus: all such $\pi$-algebras respect bisimulation congruence, and we give a concrete description of the free $\pi$-algebra $Pi(X)$ for any object $X$ of $\mathcal{S}et^{\mathcal{I}}$. We show that every free algebra is a fully-abstract model for the $\pi$-calculus, with the construction of Fiore et al. being the initial free algebra $Pi(0)$.

*E-mail address:* Ian.Stark@ed.ac.uk.

0304-3975/$ - see front matter 
doi:10.1016/j.tcs.2007.09.024

Following this, we give similar algebraic presentations for the internal mobility of $\pi$I and for the early $\pi$-calculus semantics; in each case the underlying algebraic theory is a local modification of that for the standard $\pi$-calculus.

Our method builds on a recent line of research by Plotkin and Power who use algebraic theories in enriched categories to capture "notions of computation", in particular Moggi's *computational monads* [22,32–34]. The general idea is to describe a computational feature – I/O, state, nondeterministic choice – by stating a characteristic collection of operations with specified equations between them. These then induce the following suite of constructions: a notion of algebraic model for the feature; a computational monad; effectful actions to program with; and a modal logic for specification and reasoning. This approach also gives a flexible way to express interactions between features, by combining sets of operations [14,15].

For the $\pi$-calculus, we apply and expand their technique. We take full advantage of the enriched setting, not only building models as objects in $\mathcal{S}et^{\mathcal{I}}$, but also using arities from $\mathcal{S}et^{\mathcal{I}}$ – thus our theory of $\pi$ includes operations whose arity depends on the names currently available. We use two different closed structures in $\mathcal{S}et^{\mathcal{I}}$: the cartesian exponential for standard arities, and a monoidal function space "$\multimap$" for operations parameterized by *fresh* names. Finally, the $\pi$-calculus depends on a very particular interaction between concurrency, communication and name generation, which we can directly express in equations relating the theories for each of these features. This precision in integrating different aspects of computation is a significant benefit of the algebraic approach over existing techniques for combining computational monads [16,18,23,47].

The structure of the paper is as follows. In Section 2 we review the relevant properties of algebraic theories, the $\pi$-calculus, and the functor category $\mathcal{S}et^{\mathcal{I}}$. We then set out our proposed algebraic theory of $\pi$ in Section 3. Following this, in Section 4 we show how models of the theory give a denotational semantics for the finite $\pi$-calculus (i.e., omitting recursion and replication), and prove that these interpretations respect bisimulation congruence (Proposition 2). Interestingly, parallel composition of processes is not in general admissible as a basic operation in the theory, although we are able to interpret it via expansion. We prove the existence of free algebras over $\mathcal{S}et^{\mathcal{I}}$ (Theorem 3) and show that they are all fully abstract (Theorem 5). In particular, the free algebra over the empty set is exactly the model of Fiore et al., and does support an internal definition of parallel composition (Proposition 4). We complete Section 4 by identifying the computational monad and effects induced by the theory of $\pi$, which give a programming language semantics for mobile communicating concurrency. Section 5 takes the plain theory of $\pi$ and shows how local adaptations give a theory of internal mobility in Sangiorgi's $\pi$I, and for the early semantics of $\pi$. We conclude in Section 6 by indicating possible extensions and further applications of this work.

This article is an expanded presentation of results first published in a conference paper at FOSSACS 2005 [46].

## 2. Background

### 2.1. Algebras and notions of computation

We sketch very briefly the theoretical basis for our development: for more on enriched algebraic theories see Robinson's clear and detailed exposition [38]; the link to computations and generic effects is described in [33,34].

There is a well-established connection between algebraic theories and monads on the category $\mathcal{S}et$. For example, consider the theory presented in Fig. 1, which we shall use later for an algebra $A$ of nondeterministic computations.[1] A model of this theory is a triple $\langle A, \mathit{choice}, \mathit{nil}\rangle$ of a carrier set $A$ with two maps satisfying the commuting diagrams in the figure; and these models form a category $\mathcal{ND}(\mathcal{S}et)$ of "nondeterministic sets". There is a forgetful functor $U$ from $\mathcal{ND}(\mathcal{S}et)$ to $\mathcal{S}et$, which takes an object to its carrier set. This $U$ has a left adjoint, giving the free algebra $FX$ over any set $X$.

$$\text{free}\quad F \quad \begin{array}{c} \mathcal{ND}(\mathcal{S}et) \\ \uparrow \\ \big( \dashv \big) \\ \downarrow \\ \mathcal{S}et \end{array} \quad U \quad \text{forgetful.}$$

[1] As it happens, this is also the algebraic theory of semilattices.

**Operations**

| | |
|---|---|
| $choice : A \times A \longrightarrow A$ | Combines two computations |
| $nil : 1 \longrightarrow A$ | Single deadlocked computation |

**Equations**

$$choice(p, q) = choice(q, p) \quad \text{(commutative)}$$
$$choice(p, (choice(q, r)) = choice(choice(p, q), r) \quad \text{(associative)}$$
$$choice(p, p) = p \quad \text{(idempotent)}$$
$$choice(nil, p) = p \quad \text{(unit } nil\text{)}$$

The same equations expressed as commuting diagrams:

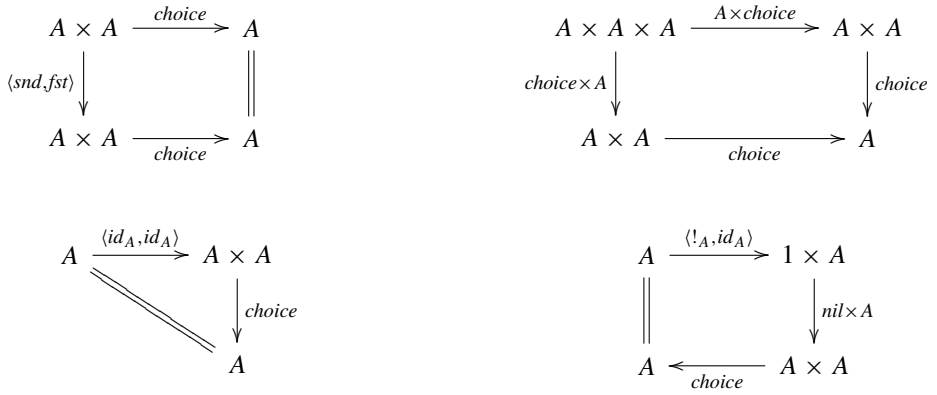



Fig. 1. Algebraic theory for nondeterministic computation.

In fact functor $F$ is the finite powerset $\langle \mathcal{P}_{fin}, \cup, \emptyset \rangle$, and $\mathcal{ND}(\mathcal{S}et)$ is *monadic* over $\mathcal{S}et$: it is equivalent to the category of algebras for the monad $\mathcal{P}_{fin}$.

The situation here is quite general, with a precise correspondence between single-sorted algebraic theories and finitary monads on $\mathcal{S}et$ (i.e., monads that preserve filtered colimits). Kelly and Power [17,36] extend this to an enriched setting: carriers for the algebras may be from some category $\mathcal{C}$ other than $\mathcal{S}et$; the arities of operations can be not just natural numbers, but certain objects in a category; and equations can be replaced with other constraint systems – for example, ordered categories support inequations.

Building on this, Plotkin and Power [34] investigate algebraic theories for Moggi-style "computational" monads $T$ [22]. Given a monad $T$, they define an *algebraic operation* to be a collection of maps $f_X : (TX)^m \to (TX)^n$ on computations where each $f_X$ and $f_Z$ commute appropriately with every $g : Y \times X \to TZ$ [34, Def. 1]. In programming language terms, this demands that operations commute with evaluation contexts. Plotkin and Power show that these algebraic operations are precisely those admissible as operations of the relevant theory, and characterize them in various enriched settings. Moreover, they prove that every algebraic operation corresponds to a computational *generic effect* of type $e_f : n \to Tm$ (note the reversal of indices $m$ and $n$). In the example above, $\mathcal{P}_{fin}$ is the standard computational monad for finite nondeterministic choice, and the effects corresponding to $choice : A^2 \to A^1$ and $nil : A^0 \to A^1$ are $\mathsf{arb} : 1 \longrightarrow \mathsf{T2}$ and $\mathsf{deadlock} : 1 \longrightarrow \mathsf{T0}$ respectively. It is well known that these two are enough to code up nondeterministic programming: $\mathsf{arb}()$ is a nondeterministic $\mathsf{true}$ or $\mathsf{false}$, and $\mathsf{deadlock}()$ is the empty choice.

Generic effects mean that not only do algebraic theories characterize computational monads as free algebras, but they also provide the necessary terms for programming with them. Algebraic theories also allow us to combine monads, a traditionally challenging area, by taking the union of theories and then selecting equations to describe how they interact [14,15].

Taken all together, these constructions lie behind Power and Plotkin's proposal of algebraic theories as a good model to capture *notions of computation* – including distinctive programming language features like state, nondeterministic choice, exceptions, and input/output.

As a second example, the theory for input/output of data values from some fixed set $V$ is:

$$in : A^V \longrightarrow A \qquad out : A \longrightarrow A^V \qquad \text{with no equations.}$$

This induces a *resumption* monad for computations performing I/O:

$$T(-) = \mu X.(X^V + V \times X + (-))$$

as well as the generic effects $\mathsf{read} : 1 \longrightarrow \mathsf{TV}$ and $\mathsf{write} : \mathsf{V} \to \mathsf{T1}$.

### 2.2. The finite $\pi$-calculus

The $\pi$-calculus presentation we use is quite standard, and here we shall just summarise notation and some definitions. For more information, consult one of the $\pi$-calculus books [20,40] or Parrow's handbook chapter [29].

Fig. 2 presents the details: process syntax, structural congruence between processes, and a small-step operational semantics in the form of an inductively defined transition relation $P \xrightarrow{\alpha} P'$. Here $x$, $y$ and $z$ range over some infinite supply of names; the prefixes $x(y).P$ and $\nu x P$ bind $y$ and $x$ respectively; and $P[z/y]$ denotes capture-avoiding substitution of $z$ for $y$ in $P$. This is a *finite* $\pi$-calculus because we omit replication or recursion in the definition of processes, so that no process can have an infinite trace.

Some presentations use a more aggressive structural congruence; for example allowing name restriction $\nu x(-)$ to change its scope. This makes no difference to the models presented here – indeed full abstraction means that we can read off these extra rules from the denotational semantics [5, §1.1].

We use a *late* transition semantics, in that input substitution happens in the (COM) rule when communication actually occurs, rather than at (IN). Section 6 discusses some ways to treat the early semantics, and variations like the internal mobility of $\pi$I.

A symmetric relation $\mathcal{S}$ between processes is said to be a *bisimulation* if for every $(P, Q) \in \mathcal{S}$ the following conditions hold:

- For $\alpha = \tau, \bar{x}y, \bar{x}(y)$, if $P \xrightarrow{\alpha} P'$ then there is $Q'$ such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{S}$.
- If $P \xrightarrow{x(y)} P'$ then there is $Q'$ such that $Q \xrightarrow{x(y)} Q'$ and for any name $z$, $(P'[z/y], Q'[z/y]) \in \mathcal{S}$.

To check this second condition it is only necessary that $z$ ranges over the free names of $P$ and $Q$, and one fresh name. The bisimulation $\mathcal{S}$ is *strong*, in that $\tau$-actions must match, and *late*, in that input actions must match before the transmitted value is known. Two processes are (strong, late) *bisimilar* if there is some bisimulation relating them. We write $P \sim Q$ and observe that bisimilarity is itself a bisimulation, and contains all others.

Bisimilarity is an equivalence relation, and is preserved by all process constructors except for input prefix $x(y).P$. This is usually proved directly from the transition semantics, but it also follows from the full abstraction of our models. The issue here with input prefix is that substitution on input may cause distinct names to become identified. In the light of this, two processes are said to be *bisimulation congruent* $P \approx Q$ if they are bisimilar under all possible name substitutions. Bisimulation congruence is then the smallest congruence containing bisimilarity.

### 2.3. The category $\mathcal{S}et^{\mathcal{I}}$

We construct our models for $\pi$ over the functor category $\mathcal{S}et^{\mathcal{I}}$, where $\mathcal{I}$ is the category of finite sets and injections. Typically we treat objects $s, s' \in \mathcal{I}$ in the index category as finite sets of names. The intuition is that an object $X \in \mathcal{S}et^{\mathcal{I}}$ is a *varying* set: if $s \in \mathcal{I}$ is the set of names available in some context, then $X(s)$ is the set of $X$-values using them. As the set of names available changes, so does this set of values; and the morphism part of $X$ describes how these values change with renaming.

Functor categories of *possible worlds* like this are well established for modelling local state in programming languages [24,26,37] and local names in particular [21,31,44]. Similar categories of varying sets also appear in models for variable binding [7] and name binding (see, for example, [41] and citations there).

In fact, all of the functors we shall need in $\mathcal{S}et^{\mathcal{I}}$ are *pullback-preserving* and our constructions all preserve this property, so we could as well work in the full subcategory $\mathcal{A}$ of pullback-preserving functors from $\mathcal{I}$ to $\mathcal{S}et$. From the viewpoint of varying sets, $X$ is in $\mathcal{A}$ precisely when every $x \in X(s)$ has some unique least $s_0 \subseteq s$ with

**Syntax**

Processes

| | | | | | | |
|---|---|---|---|---|---|---|
| $P, Q$ | $::=$ | $\bar{x}y.P$ | output | $\mid$ | $0$ | deadlock |
| | $\mid$ | $x(y).P$ | input | $\mid$ | $P + Q$ | nondeterministic choice |
| | $\mid$ | $\nu x P$ | restriction | $\mid$ | $P \mid Q$ | parallel composition |

**Structural congruence**

The least equivalence relation closed under term constructors and containing:

$$P + 0 \equiv P \qquad x(y).P \equiv x(z).P[z/y] \quad z \notin fn(P)$$
$$P \mid 0 \equiv P \qquad \nu y P \equiv \nu z P[z/y] \quad z \notin fn(P)$$
$$P + Q \equiv Q + P \qquad (P + Q) + R \equiv P + (Q + R)$$
$$P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$$

**Actions**

| | | | | | | |
|---|---|---|---|---|---|---|
| $\alpha$ | $::=$ | $\tau$ | internal | $\mid$ | $\bar{x}y$ | free output |
| | $\mid$ | $x(y)$ | input | $\mid$ | $\bar{x}(y)$ | bound output |

**Transition rules**

$$\text{OUT}\quad \bar{x}y.P \xrightarrow{\bar{x}y} P \qquad \text{IN}\quad x(y).P \xrightarrow{x(y)} P$$

$$\text{PAR}\quad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ bn(\alpha) \not\subseteq fn(Q) \qquad \text{SUM}\quad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$\text{RES}\quad \frac{P \xrightarrow{\alpha} P'}{\nu x P \xrightarrow{\alpha} \nu x P'}\ x \notin fn(\alpha) \qquad \text{COM}\quad \frac{P \xrightarrow{x(y)} P' \quad Q \xrightarrow{\bar{x}z} Q'}{P \mid Q \xrightarrow{\tau} P'[z/y] \mid Q'}$$

$$\text{OPEN}\quad \frac{P \xrightarrow{\bar{x}y} P'}{\nu y P \xrightarrow{\bar{x}(y)} P'}\ x \neq y \qquad \text{CLOSE}\quad \frac{P \xrightarrow{x(y)} P' \quad Q \xrightarrow{\bar{x}(y)} Q'}{P \mid Q \xrightarrow{\tau} \nu y(P' \mid Q')}$$

$$\text{STRUCT}\quad \frac{P \equiv Q \quad P \xrightarrow{\alpha} P' \quad P' \equiv Q'}{Q \xrightarrow{\alpha} Q'}$$

Fig. 2. The finite $\pi$-calculus with late transition semantics.

$x_0 \in X(s_0)$ a preimage of $x$; that is, each element depends on some well-defined least set of names. Conveniently, pullback-preserving functors also preserve monomorphisms; and every map in $\mathcal{I}$ is mono, so all the base maps $X(f) : X(s) \to X(s')$ in $\mathcal{S}et$ that we shall meet are injective.

The category $\mathcal{A}$ has other characterisations, in particular as the *Schanuel topos* of sheaves for the atomic topology on $\mathcal{I}^{\mathrm{op}}$ [19, pp. 115, 155, 158] and as a category of sets which are acted on by permutations of the natural numbers $\mathbb{N}$ [19, pp. 150–155]. This last form demonstrates $\mathcal{A}$ as a permutation model for Fraenkel–Mostowski set theory with atoms, which Pitts and Gabbay have very successfully used as the basis of their *nominal sets* approach to reasoning about names and binding [11,30].

Setting aside these intriguing connections, we opt for simplicity and work primarily with $\mathcal{S}et^{\mathcal{I}}$ in the development to follow. There it plays two distinct roles. First, it is the arena within which we build name-sensitive algebras and monads. Second, it is also the source of arities for algebraic operations: in particular, our input and output operations

have arities that vary depending on the names available. The remainder of this section sets out all the properties of $\mathcal{S}et^{\mathcal{I}}$ that we use to support these two roles.

Category $\mathcal{S}et^{\mathcal{I}}$ is complete and cocomplete, with limits and colimits taken pointwise. It is cartesian closed, with a convenient way to calculate function spaces using natural transformations between functors:

$$X \times Y \qquad (X \times Y)(s) = X(s) \times Y(s)$$
$$X \to Y \text{ or } Y^X \qquad Y^X(s) = \mathcal{S}et^{\mathcal{I}}[X(s + \_), Y(s + \_)].$$

Thus elements in the varying set of functions from $X$ to $Y$ over names $s$ must take account of values in $X(s + s')$, uniformly for all extended name sets $s + s'$.

There is also a symmetric monoidal closed structure $(\otimes, \multimap)$ around the *Day tensor* [6], induced by disjoint union $(s + s')$ in $\mathcal{I}$.

$$X \otimes Y = \int^{s,s' \in \mathcal{I}} X(s) \times Y(s') \times \mathcal{I}[s + s', \_].$$

For those functors that preserve pullbacks we can give an explicit presentation of the monoidal structure:

$$(X \otimes Y)(s) = \left\{(x, y) \in (X \times Y)(s) \;\middle|\; \exists \text{disjoint } s_1, s_2 \subseteq s \;.\; x \in X(s_1), y \in Y(s_2)\right\}$$
$$(X \multimap Y)(s) = \mathcal{S}et^{\mathcal{I}}[X(\_), Y(s + \_)].$$

Elements of $(X \otimes Y)$ denote pairs of elements from $X$ and $Y$ that use disjoint name sets. Elements of the monoidal function space $(X \multimap Y)$ are functions defined only at $X$-values that use just fresh names.

The two closed structures are related:

$$into_{X,Y} : X \otimes Y \longrightarrow X \times Y$$
$$onto_{X,Y} : (X \to Y) \longrightarrow (X \multimap Y).$$

Where the functors $X$ and $Y$ are pullback-preserving, these are an inclusion and surjection, respectively. We shall also use the following distributive laws:

$$dist_{\otimes/\times} : X \otimes (Y \times Z) \longrightarrow Y \times (X \otimes Z)$$
$$dist_{\multimap/\to} : (X \multimap (Y \to Z)) \longrightarrow (Y \to (X \multimap Z)).$$

The cartesian and monoidal structures share the same unit, 1.

We use a variety of objects in $\mathcal{S}et^{\mathcal{I}}$. For any fixed set $S$, there is a corresponding constant functor $S \in \mathcal{S}et^{\mathcal{I}}$. The *object of names* $N \in \mathcal{S}et^{\mathcal{I}}$ is the inclusion functor mapping any $s \in \mathcal{I}$ to the same $s \in \mathcal{S}et$. From this we build $(N \times N \times \cdots \times N) = N^k$, the object of $k$-tuples of names, and $(N \otimes N \otimes \cdots \otimes N) = N^{\otimes k}$ of distinct $k$-tuples, with an inclusion $into_k : N^{\otimes k} \hookrightarrow N^k$ between them.

We have the *shift* functor $\delta$ on objects of $\mathcal{S}et^{\mathcal{I}}$:

$$\delta : \mathcal{S}et^{\mathcal{I}} \longrightarrow \mathcal{S}et^{\mathcal{I}} \quad \text{defined by} \quad \delta X(\_) = X(\_ + 1).$$

In fact $\delta(-) \cong N \multimap (-)$, and elements of $\delta X$ are elements of $X$ that may use a single fresh name, uniformly in the choice of that name. Shift interacts smoothly with other constructions:

$$\delta(X \times Y) \cong \delta X \times \delta Y \qquad \delta(X \to Y) \cong \delta X \to \delta Y \qquad \delta N \cong (N + 1).$$

The functor $\delta$ is well known, for example as *dynamic allocation* in [8,9]; it also appears as the *atom abstraction* operator $[N]X$ of FM-set theory identified by Gabbay and Pitts [11,30].

The representable objects in $\mathcal{S}et^{\mathcal{I}}$ are 1, $N$, $(N \otimes N)$, $(N \otimes N \otimes N)$, .... The finitely presentable (finitary) objects are the finite colimits of these, including in particular finite constant sets $S$, and all finite products of $N$: for example, $(N \times N) \cong N + (N \otimes N)$. These finitely presentable objects are the ones available as arities for algebraic theories over $\mathcal{S}et^{\mathcal{I}}$.

Finally, the category $\mathcal{S}et^{\mathcal{I}}$ is locally finitely presentable as a closed category, with respect to both cartesian and monoidal structures. This is a completeness requirement for building algebraic theories; it gives us that 1, $\times$ and $\otimes$ all restrict to the subcategory of finitely presentable objects, and that all objects are filtered colimits of finitely presentable ones. For further details on what this involves see [36, §2] and [38, §3].

## 3. Theory of $\pi$

The algebraic approach supports a modular presentation of theories, and we use this to manage the combination of features that come together in the $\pi$-calculus. This section presents in turn separate theories for nondeterministic choice, communication along channels, and dynamic name creation; followed by equations specifying exactly how these features should interact.

We assume a carrier object $A \in \mathcal{S}et^{\mathcal{I}}$, and describe the operations and equations required for $A$ to model the $\pi$-calculus.

### 3.1. Nondeterministic choice

We have already seen the appropriate algebraic theory for nondeterministic computation, in Fig. 1. The statement here is the same, only now over $\mathcal{S}et^{\mathcal{I}}$ rather than $\mathcal{S}et$: we need a binary *choice* operation which is commutative, associative and idempotent with a unit *nil*.

| | |
|---|---|
| $choice : A^2 \longrightarrow A$ | $choice(p, q) = choice(q, p)$ |
| $nil : 1 \longrightarrow A$ | $choice(nil, p) = choice(p, p) = p$ |
| | $choice(p, (choice(q, r)) = choice(choice(p, q), r)$ |

In process calculus terms, *choice* captures nondeterministic sum $P + Q$ and *nil* the deadlocked process 0.

### 3.2. Communication

Communication in the $\pi$-calculus is along named channels, sending names themselves as data. The relevant theory is a specialised version of that for I/O given earlier.

| | |
|---|---|
| $out : A \longrightarrow A^{N\times N}$ | |
| $in : A^N \longrightarrow A^N$ | (No required equations) |
| $tau : A \longrightarrow A$ | |

These three operations correspond to the three prefixing constructions of the $\pi$-calculus: output $\bar{x}y.P$, input $x(y).P$ and internal action $\tau.P$. Argument and result arities follow the bound and free occurrences of names respectively:

- *out* is parameterized in the result $A^{N\times N}$ by both channel and data names;
- *in* accepts argument $A^N$ parameterized by the data value, with result $A^N$ parameterized by channel name.

The appearance of $A^N$ and $A^{N\times N}$ here give our first nonstandard arities, $N$ and $N \times N$, to describe operations whose arity varies according to the names currently available. We shall follow [33] in using formal indices to write these down: with terms like $out_{x,y}(p)$ and $in_x(q_y)$, where $x$ and $y$ are name parameters. Notice that this parameter notation applies equally to operations and their arguments: so $out_{x,y}(p)$ is an $N \times N$-indexed collection of *out* operations, each taking a single argument $p$; while $in_x(q_y)$ is an $N$-indexed collection of *in* operations, each taking an $N$-indexed argument collection $q_y$.

### 3.3. Dynamic name creation

Processes in the $\pi$-calculus can dynamically generate fresh communication channels: term $\nu x P$ is the process that creates a new channel, binds it to the name $x$, and then becomes process $P$ which may then use the new channel.

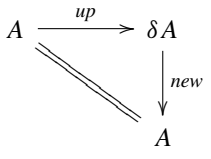
Our theory for this is a modification of Plotkin and Power's *block* operation for local state [33, §4]. We require a single operation *new* with a monoidal arity.

| | |
|---|---|
| $new : \delta A \longrightarrow A$ | $new(x.p) = p$ for $p$ independent of $x$ |
| | $new(x.new(y.p)) = new(y.new(x.p))$ |

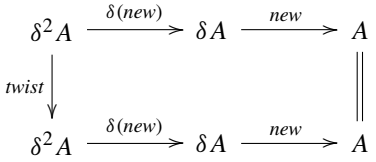
The argument $\delta A$ means that *new* is an operation of arity $N$ in the monoidal closed structure of $\mathcal{S}et^{\mathcal{I}}$; remembering from Section 2.3 that $\delta A \cong (N \multimap A)$.

To express equations over this monoidal arity, recall that elements of $\delta A$ are elements of $A$ which depend on a single fresh name, uniformly in the choice of that fresh name. We write such an element as $x.p$, for the term $p$ indexed by fresh $x$, borrowing Gabbay and Pitts's notation for atom abstraction [11]. (Plotkin and Power write this as $\langle p \rangle_x$.)

Strictly speaking, all our equations are shorthand for certain diagrams in $\mathcal{S}et^{\mathcal{I}}$ which must commute. These two state that the creation of unused fresh names cannot be observed, and computation is independent of the order in which fresh names are created. In diagram form, these are




and




where $up : 1 \to \delta$ and $twist : \delta^2 \to \delta^2$ are the evident natural transformations on the shift functor.

At first sight, it might seem that a more conventional cartesian arity $A^N \to A$ would suffice for *new*, and that we could dispense entirely with the monoidal structure of $\mathcal{S}et^{\mathcal{I}}$. However, this is not so: Section 3.5 below makes essential use of monoidal arities to describe the interaction of name creation with other component theories. In particular the equations combining *new* with *in* and *out* depend on the inequality of certain name parameters, and we capture this in a commuting diagram with both $N \multimap (-)$ and $N \to (-)$ arities.

### 3.4. Other operations

There are a few further constructions that we might expect as candidates for inclusion in a theory of $\pi$.

**Name testing.** Some forms of the $\pi$-calculus allow direct comparison of names, with prefixes like match $[x = y]P$, mismatch $[x \neq y]Q$, or two-branched testing $(x = y)\ ?\ P : Q$. It turns out that these operations are already in the theory. In Section 2.3 we have already seen the isomorphism of arities $(N \times N) \cong (N + (N \otimes N))$ in $\mathcal{S}et^{\mathcal{I}}$. From this we get:

$$N \times N \xrightarrow{\ \cong\ } N + (N \otimes N) \xrightarrow{\ !_N + !_{N \otimes N}\ } 1 + 1$$

which lifts to a testing operation, available in any theory over $\mathcal{S}et^{\mathcal{I}}$:

$$test : A^2 \longrightarrow A^{N \times N} \qquad \begin{aligned} test_{x,x}(p,q) &= p \\ test_{x,y}(p,q) &= q \quad x \neq y. \end{aligned}$$

From this and *nil* we can define operations for match and mismatch.

$$eq : A \longrightarrow A^{N \times N} \qquad eq_{x,y}(p) \stackrel{def}{=} test_{x,y}(p, nil)$$

$$neq : A \longrightarrow A^{N \times N} \qquad neq_{x,y}(p) \stackrel{def}{=} test_{x,y}(nil, q).$$

**Bound output.** The bound output prefix $\bar{x}(y).P$ for the $\pi$-calculus is equivalent to $\nu y(\bar{x}y.P)$. There is an analogous derived operation in the theory:

$$bout : \delta A \longrightarrow A^N \qquad bout_x(y.p) \stackrel{def}{=} new(y.out_{x,y}(p)).$$

Because this is definable in terms of the operations given earlier, it can be included without affecting the induced theory or its algebras.

**Parallel composition.** The usual process calculus construction $(P \mid Q)$ is not directly admissible as an operation in our theory of $\pi$. This is because it is not *algebraic* in the sense of Plotkin and Power, described earlier in Section 2.1 and in [34, Defn. 1]. Specifically, parallel composition does not commute with evaluation contexts: for example, in a programming language the sequence $(\mathsf{M} \mid \mathsf{M}'); \mathsf{N}$ is not in general equivalent to $(\mathsf{M}; \mathsf{N}) \mid (\mathsf{M}'; \mathsf{N})$.

Although parallel composition cannot be written into our theory of $\pi$, we can still capture $(P \mid Q)$ in all models of the theory; we shall see more on this later, in Section 4.

### 3.5. Combining equations

To complete the theory of $\pi$ we also need equations that specify how the component theories interact. The algebraic approach gives us some flexibility in doing so, as investigated in [14,15]. For example, we can assert no additional equations, giving the *sum* of theories [15, §3]; we can require that the operations from two theories commute with each other, to give the commutative combination, or *tensor*, of theories [15, §4]; or we can choose some other custom interaction. To assemble the component theories of $\pi$, we use all three methods:

- The sum of the theories of nondeterministic choice and communication.
- The commuting combination of nondeterministic choice and name creation.
- A custom set of equations for name creation and communication; mostly commuting, but with some specific interaction.

These expand into three sets of equations. The first have effect by their absence:

**Sum of component theories**

No equations required for *choice* or *nil* with *out*, *in* or *tau*.

The commuting combination of theories says that operations act independently:

**Commuting component theories**

$$new(x.choice(p,q)) = choice(new(x.p), new(x.q))$$

$$new(z.out_{x,y}(p)) = out_{x,y}(new(z.p)) \qquad z \notin \{x,y\}$$

$$new(z.in_x(p_y)) = in_x(new(z.p)_y) \qquad z \notin \{x,y\}$$

$$new(z.tau(p)) = tau(new(z.p))$$

As before, these equations with formal indices and side conditions are a shorthand for commuting diagrams in $\mathcal{S}et^{\mathcal{I}}$; see Fig. 3. In particular, the side conditions requiring freshness of $z$ show up as uses of the monoidal exponential, in the form of $\delta(-)$. As noted earlier, this is the key point where we require a monoidal rather than cartesian arity for *new*.

Another way to see these side conditions is to consider the implicit quantification of the parameters $p$, $q$, $x$, $y$ and $z$. In a conventional cartesian setting these are all universally quantified; but with the additional nominal structure of $\mathcal{S}et^{\mathcal{I}}$, there is the possibility of either the universal $\forall$ or the fresh $\mathit{N}$ of Pitts. What is more, ordering then becomes important: $\forall x \mathit{N} y \ldots$ ensures that $y \neq x$, while $\mathit{N} y \forall x \ldots$ does not. Here we could eliminate side conditions by adding explicit quantification $\forall x \forall y \mathit{N} z \forall p$ to both the second and third equations.

Finally, just two equations for interaction capture the precise flavour of the $\pi$-calculus: that the binder $\nu x(-)$ is both creation (of new channels) and restriction (of communication on them).

**Interaction between component theories**

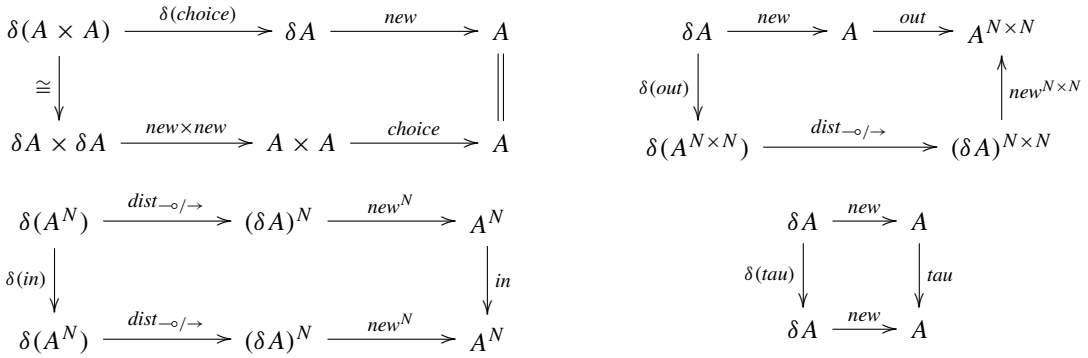
$$new(x.out_{x,y}(p)) = nil$$

$$new(x.in_x(p_y)) = nil$$

Notice that by relating *nil*, *in*/*out*, and *new*, these equations connect all three component theories at once. Fig. 3 expands them into commuting diagrams.

Fig. 4 gathers together the operations and equations from the preceding sections to summarise the algebraic theory of (strong, late) $\pi$.

**Commuting component theories**




**Interaction between component theories**

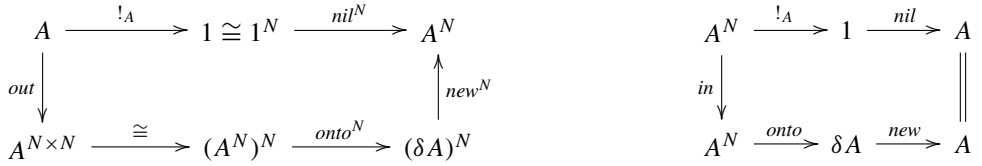



Fig. 3. Commuting diagrams for combining component theories.

**Operations**

$$choice : A^2 \longrightarrow A \qquad out : A \longrightarrow A^{N\times N} \qquad tau : A \longrightarrow A$$
$$nil : 1 \longrightarrow A \qquad in : A^N \longrightarrow A^N \qquad new : \delta A \longrightarrow A$$

**Equations**

$$choice(p, q) = choice(q, p)$$
$$choice(nil, p) = choice(p, p) = p$$
$$choice(p, (choice(q, r)) = choice(choice(p, q), r)$$

$$new(x.p) = p \qquad x \text{ fresh for } p$$
$$new(x.new(y.p)) = new(y.new(x.p))$$

$$new(x.choice(p, q)) = choice(new(x.p), new(x.q))$$

$$new(z.out_{x,y}(p)) = out_{x,y}(new(z.p)) \qquad z \notin \{x, y\}$$
$$new(z.in_x(p_y)) = in_x(new(z.p_y)) \qquad z \notin \{x, y\}$$
$$new(z.tau(p)) = tau(new(z.p))$$

$$new(x.out_{x,y}(p)) = nil$$
$$new(x.in_x(p_y)) = nil$$

Fig. 4. Operations and equations for a theory of $\pi$.

## 4. Algebraic models for $\pi$

We now turn to look at models for the theory of $\pi$. We define what these are, and show that every such model gives a denotational semantics for the $\pi$-calculus that respects bisimulation congruence. We give a construction for free

models in $\mathcal{S}et^{\mathcal{I}}$, and prove that the category of models is monadic over $\mathcal{S}et^{\mathcal{I}}$. We show that all free models are fully abstract for bisimulation congruence, and in particular that the initial free model is isomorphic to the construction of Fiore et al..

### 4.1. Categories of algebras

**Definition 1.** A $\pi$-*algebra* in $\mathcal{S}et^{\mathcal{I}}$ is an object $A$ together with maps (*choice*, *nil*, *out*, *in*, *tau*, *new*) satisfying the equations in Fig. 4. These algebras form a category $\mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$, with morphisms the maps $f : A \to B$ that commute with all operations. The forgetful functor $U : \mathcal{PI}(\mathcal{S}et^{\mathcal{I}}) \to \mathcal{S}et^{\mathcal{I}}$ takes a $\pi$-algebra to its carrier object.

For any $\pi$-algebra $A \in \mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$ we can build a denotational semantics of the finite $\pi$-calculus: if $P$ is a process with free names in set $s$, then there is a map

$$[\![s \vdash P]\!]_A : N^{|s|} \longrightarrow A.$$

Here $N^{|s|}$ represents an environment instantiating the free names $s$.

The interpretation itself is comparatively straightforward, by induction over the syntactic structure of process terms; the details are in Fig. 5. Process sum, nil and the $\pi$-calculus prefixes are interpreted directly by the corresponding $\pi$-algebra operations. Binding of fresh names involves managing the monoidal structure; we use a construction $\nu(-)$ on maps into $A$:

$$\begin{aligned}
p : N^{|s|+1} &\longrightarrow A && \text{Given a map } p;\\
N \otimes N^{|s|} \xrightarrow{into} N \times N^{|s|} &\longrightarrow A && \text{precompose inclusion;}\\
N^{|s|} &\longrightarrow (N \multimap A) && \text{take the monoidal transpose;}\\
N^{|s|} &\longrightarrow \delta A \xrightarrow{new} A && \text{and apply the } new \text{ operator}\\
\nu p : N^{|s|} &\longrightarrow A && \text{to get the restricted map } \nu p.
\end{aligned}$$

We then define $[\![s \vdash \nu x\, P]\!]_A = \nu([\![s, x \vdash P]\!]_A)$.

As noted earlier, parallel composition is not algebraic, so we have no general map for its action on $A$. However, for any specific finite processes $P$ and $Q$ we can use the expansion law for congruence [29, Table 9] to express $(P \mid Q)$ as a sum of smaller processes, and so obtain an interpretation in the $\pi$-algebra $A$, recursively:

$$\text{if} \qquad P \mid Q = \sum_{i=1}^{k} R_i \qquad \text{(canonical choice of expansion)}$$

$$\text{then} \qquad [\![s \vdash P \mid Q]\!]_A = \mathit{choice}([\![s \vdash R_1]\!]_A, \mathit{choice}([\![s \vdash R_2]\!]_A, \ldots)) : N^{|s|} \longrightarrow A.$$

This external expansion makes the translation not wholly compositional, which is somewhat unsatisfactory. We shall revisit this in Section 4.3 and improve the situation, for certain $\pi$-algebras, by expressing parallel composition within the algebra itself.

The interpretation $[\![s \vdash P]\!]_A$ respects weakening of the name context $s$, so we usually omit it and write $[\![P]\!]_A$.

Once defined, this interpretation induces a notion of equality over a model: for any $\pi$-algebra $A$ and finite processes $P$, $Q$ we write

$$A \models P = Q \quad \overset{def}{\Longleftrightarrow} \quad [\![P]\!]_A = [\![Q]\!]_A$$

$$\text{and} \quad \mathcal{S}et^{\mathcal{I}} \models P = Q \quad \overset{def}{\Longleftrightarrow} \quad A \models P = Q \quad \text{for all } A \in \mathcal{PI}(\mathcal{S}et^{\mathcal{I}}).$$

**Proposition 2.** *All $\pi$-algebra models respect (strong, late) bisimulation congruence. For any $A \in \mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$ and finite processes $P$, $Q$:*

$$P \approx Q \quad \Longrightarrow \quad A \models P = Q$$

*and more generally:*

$$P \approx Q \quad \Longrightarrow \quad \mathcal{S}et^{\mathcal{I}} \models P = Q.$$

Denotational semantics for $\pi$-calculus terms:

$$[\![s \vdash \bar{x}y.P]\!] = out_{x,y}([\![s \vdash P]\!]) \qquad [\![s \vdash \nu x P]\!] = \nu(x.[\![s, x \vdash P]\!])$$
$$[\![s \vdash x(y).P]\!] = in_x([\![s, y \vdash P]\!]) \qquad [\![s \vdash 0]\!] = nil$$
$$[\![s \vdash P + Q]\!] = choice([\![s \vdash P]\!], [\![s \vdash Q]\!])$$
$$[\![s \vdash P \mid Q]\!] = choice([\![s \vdash R_1]\!], choice([\![s \vdash R_2]\!], \ldots))$$

where $P \mid Q = \sum_{i=1}^{k} R_i$, canonical choice of expansion.

The same, with morphisms presented diagrammatically:

$$\frac{[\![s \vdash P]\!] : N^{|s|} \longrightarrow A}{[\![s \vdash \bar{x}y.P]\!] : N^{|s|} \xrightarrow{\langle [\![s \vdash P]\!], \langle proj_x, proj_y \rangle \rangle} A \times N^2 \xrightarrow{out \times N^2} A^{N^2} \times N^2 \xrightarrow{ev} A}$$

$$\frac{[\![s, y \vdash P]\!] : N^{|s|} \times N \longrightarrow A}{[\![s \vdash x(y).P]\!] : N^{|s|} \xrightarrow{\langle \ulcorner [\![s, y \vdash P]\!] \urcorner, proj_x \rangle} A^N \times N \xrightarrow{in \times N} A^N \times N \xrightarrow{ev} A}$$

$$\frac{[\![s, x \vdash P]\!] : N^{|s|} \times N \longrightarrow A}{[\![s \vdash \nu x P]\!] : N^{|s|} \xrightarrow{\ulcorner [\![s, x \vdash P]\!] \circ into \urcorner} \delta A \xrightarrow{new} A}$$

$$[\![s \vdash 0]\!] : N^{|s|} \xrightarrow{!} 1 \xrightarrow{nil} A$$

$$\frac{[\![s \vdash P]\!] : N^{|s|} \longrightarrow A \qquad [\![s \vdash Q]\!] : N^{|s|} \longrightarrow A}{[\![s \vdash P + Q]\!] : N^{|s|} \xrightarrow{\langle [\![s \vdash P]\!], [\![s \vdash Q]\!] \rangle} A \times A \xrightarrow{choice} A}$$

$$\frac{[\![s \vdash R_i]\!] : N^{|s|} \longrightarrow A \qquad P \mid Q = \sum_{i=1}^{k} R_i}{[\![s \vdash P \mid Q]\!] : N^{|s|} \xrightarrow{\langle [\![s \vdash R_i]\!] \rangle_i} A^k \xrightarrow{choice} \cdots \xrightarrow{choice} A}$$

Fig. 5. Interpretation of $\pi$-calculus terms in an arbitrary $\pi$-algebra $A$.

**Proof.** We draw on the known axiomatization of bisimulation congruence for finite processes, as given for example in [29, §8.2]. All these axioms are provable in the theory of $\pi$ and hence hold in every algebra for the theory. □

The same interpretation can also capture bisimilarity $P \sim Q$ rather than bisimulation congruence, for any $\pi$-algebra $A$. Replace the environment of names $N^{|s|}$ with the object of distinct names $N^{\otimes|s|}$, and prefix with the inclusion $N^{\otimes|s|} \hookrightarrow N^{|s|}$, to get maps that represent process behaviour up to bisimilarity:

$$(\!|s \vdash P|\!)_A : N^{\otimes|s|} \xrightarrow{into} N^{|s|} \xrightarrow{[\![s \vdash P]\!]_A} A\,.$$

Analogues of Proposition 2, and later full abstraction, follow for these maps too.

Between $N^{\otimes|s|}$ and $N^{|s|}$ lies a spectrum of $s$-fold name objects, representing sets of names up to *distinctions*, or more generally *conditions*: constraints on which free names may or may not become identified. For each of these there is a matching notion of bisimilarity-up-to, similarly captured by an interpretation based on prefixing $[\![P]\!]_A$ with an injection into $N^{|s|}$.

### 4.2. Free $\pi$-algebras in $\mathcal{S}et^{\mathcal{I}}$

The previous section proposes a theory of algebraic models for the $\pi$-calculus; but it does not yet give us any concrete $\pi$-algebras. For these we seek a free $\pi$-algebra functor $F : \mathcal{S}et^{\mathcal{I}} \to \mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$, left adjoint to the forgetful $U$. Kelly and Power [17,36] show the existence in general of such algebras for enriched theories; but there are two difficulties in our situation. First, their results are in terms of a general colimit, and for any specific theory one would also like a direct form if possible. Second, and more serious, they treat a single enrichment, while we have two together.

We can overcome both of these difficulties, in the specific case of $\mathcal{S}et^{\mathcal{I}}$: we have an explicit description of the free $\pi$-algebras, and an accompanying proof that they are so.

Before presenting the free algebras for the full theory of $\pi$, we detour briefly through those for each of its component theories, to see how they fit together. For simplicity we present not the free functors $F$, but only the associated monads $T = (U \circ F)$ on $\mathcal{S}et^{\mathcal{I}}$.

The monad for finite nondeterministic choice is the finite covariant powerset, extended pointwise to $\mathcal{S}et^{\mathcal{I}}$:

$$T_{nondet}(-) = \mathcal{P}_{fin}(-).$$

The monad for communication is a version of the resumption monad from the introduction, with components for output, input and silent internal action:

$$T_{comm}(-) = \mu Y.(N \times N \times Y + N \times Y^N + Y + (-)).$$

Here $\mu Y.(-)$ is the least fixed point, which in $\mathcal{S}et^{\mathcal{I}}$ is a straightforward pointwise union. Informally, an element of $(T_{comm}(X))(s)$ is a finite trace of $\pi$-calculus actions using names from $s$, finishing with a value from $X$; with the refinement that at input actions the function space $Y^N$ gives a branching over possible input names, including uniform treatment of new names.

The monad for dynamic name creation is that originating with Moggi [21, §4.1.4] and investigated in [44].

$$T_{new}(-) = \mathcal{D}yn(-) = \int^{s \in \mathcal{I}} \delta^{|s|}(-).$$

This is a colimit over possible sets of fresh names. In particular, the object part has $\mathcal{D}yn(X)(s) = \sum_{s' \in \mathcal{I}} X(s+s')/\sim$, where $\sim$ is an equivalence relation generated by injections between fresh name sets $s' \rightarrowtail s''$. For full element-by-element details of the $\mathcal{D}yn$ construction, see [44, §5].

Taking the approach of combining monads through *monad transformers* [18], we can try to interleave these to obtain a candidate monad for $\pi$:

$$T_{bad}(-) = \mu Y.(\mathcal{P}_{fin}(\mathcal{D}yn(N \times N \times Y + N \times Y^N + Y + (-)))).$$

Working from the outside in, this asserts that: a $\pi$-calculus process is a recursive system ($\mu Y$); which may have several courses of action ($\mathcal{P}_{fin}$); that each may create fresh names ($\mathcal{D}yn$); and then perform some I/O action, to give some further process.

However, this is not yet quite right: $T_{bad}$ does not validate any of the equations of Section 3.5 for combining the different $\pi$-calculus effects. For example, in $T_{bad}$ restriction *new* does not commute with *choice*; nor does it in fact restrict, as there are terms in the monad for external I/O on a *new*-bound channel.

To find the correct monad for $\pi$, we use an observation from existing operational treatments: name creation is only observable through the emission of fresh names in bound output. This leads to the following corrected definition:

$$T_\pi(-) = \mu Y.(\mathcal{P}_{fin}(N \times N \times Y + N \times \delta Y + N \times Y^N + Y + \mathcal{D}yn(-))). \tag{1}$$

This still expresses a $\pi$-calculus process as a recursive system ($\mu Y$) with several courses of action ($\mathcal{P}_{fin}$); but the global application of $\mathcal{D}yn(-)$ has been replaced by a bound output term $N \times \delta Y$ in the I/O expression. The core expression now contains terms for free output, bound output, input, and internal action; and matches the functor $H$ of Fiore et al. [10, §4.4].

More concretely, for any object $X$ and finite name set $s$ the set $T_\pi(X)(s)$ is (isomorphic to) the set of $V$-normal forms [10, §2.2] for all $\pi$-calculus terms built using the names in $s$; and for any injective renaming $f : s \rightarrowtail s'$ the map $T_\pi(X)(f) : T_\pi(X)(s) \to T_\pi(X)(s')$ carries out name substitution on terms.

The monad $T_\pi$ is now a correct representation for $\pi$-calculus behaviour, and for any object $X \in \mathcal{S}et^{\mathcal{I}}$ we can equip $T_\pi(X)$ with the six required operations to make it a $\pi$-algebra $Pi(X)$. The interesting case is *new*; this is defined recursively by cases, using the equations from Sections 3.3 and 3.5, and following the pattern of [45, §3.2 & Fig. 2] and [10, Table 4].

We thus obtain the desired free functor $Pi : \mathcal{S}et^{\mathcal{I}} \to \mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$, and hence a supply of concrete $\pi$-algebras. This completes the adjunction $Pi \dashv U$, with monad $U \circ Pi$ being $T_\pi$. What is more, the adjunction is monadic, so that $\mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$ is equivalent to the category of algebras for the monad $T_\pi$. To summarise:

**Theorem 3.** (i) *The forgetful functor $U : \mathcal{PI}(\mathcal{S}et^{\mathcal{I}}) \to \mathcal{S}et^{\mathcal{I}}$ has a left adjoint Pi giving a free $\pi$-algebra $Pi(X)$ over any $X \in \mathcal{S}et^{\mathcal{I}}$.*

(ii) *The comparison functor from $\mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$ to $T_\pi$-Alg is an equivalence of categories.*

**Proof.** (i) Once we have an explicit form for $Pi$, it only remains to check that $Pi(X)$ is initial among $\pi$-algebras over $X$. Given any $\pi$-algebra $A$ with $X \to UA$ in $\mathcal{S}et^{\mathcal{I}}$, we must extend this to an algebra map $Pi(X) \to A$. The extension is uniquely determined by the fact that every element of $Pi(X)$ can be generated from $X$ using operations from the theory of $\pi$.

(ii) The explicit form for $T_\pi$ allows us to show the equivalence directly on objects. The usual comparison functor recognises that for any $A \in \mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$ the six $\pi$-algebra operations are sufficient to build a map $T_\pi(UA) \to UA$, putting a $T_\pi$-algebra structure on the carrier of $A$. Conversely, given any $T_\pi$-algebra $h : T_\pi X \to X \in \mathcal{S}et^{\mathcal{I}}$ we can identify a $\pi$-algebra over carrier $X$. We extract the necessary operations from the structure map $h$ by unrolling the recursion in $T_\pi$ and picking apart the core coproduct. For example, $out_X$ is the transpose of the following composition:

$$N \times N \times X \xrightarrow{N^2 \times \eta_X} N \times N \times T_\pi X \xrightarrow{\{\!|\cdot|\!\} \circ in} \mathcal{P}_{fin}(N \times N \times T_\pi X + \cdots) \cong T_\pi X \xrightarrow{h} X.$$

These constructions are inverse to each other, up to isomorphism, and give the necessary equivalence of categories.

Alternatively, we can obtain the same result by taking a proof of Power and adapting it to handle the two simultaneous closed structures used in the theory of $\pi$ [36, Thm. 4.2]. This applies Beck's theorem to show that the adjunction $Pi \dashv U$ is monadic, and does not make use of the explicit structure of $Pi$. □

### 4.3. Parallel composition

The interpretation in Section 4.1 of $\pi$-calculus terms in an arbitrary $\pi$-algebra is not altogether compositional, in that we expand out parallel processes first. For free $\pi$-algebras we can do a little better.

**Proposition 4.** *For any pair of free $\pi$-algebras $Pi(X)$ and $Pi(Y)$ there is a map on their carrier objects:*

$$par_{X,Y} : Pi(X) \times Pi(Y) \longrightarrow Pi(X \times Y) \qquad \text{in } \mathcal{S}et^{\mathcal{I}}$$

*such that for all finite $\pi$-calculus processes $P$, $Q$:*

$$[\![P \mid Q]\!]_{Pi(X \times Y)} = par_{X,Y}([\![P]\!]_{Pi(X)}, [\![Q]\!]_{Pi(Y)}).$$

*Furthermore, for any free $\pi$-algebra $Pi(X)$ and associative-commutative map $\mu : X \times X \to X$ there is a map*

$$par_\mu : Pi(X) \times Pi(X) \longrightarrow Pi(X)$$

*such that for all finite $\pi$-calculus processes $P$, $Q$:*

$$[\![P \mid Q]\!]_{Pi(X)} = par_\mu([\![P]\!]_{Pi(X)}, [\![Q]\!]_{Pi(X)}).$$

**Proof.** We decompose $par_{X,Y}$ as a sum of communication and interleaving left merge, and then define each of these recursively by cases on the expansion of $Pi(X)$ and $Pi(Y)$. This is the procedure known from existing models, as in

[45, §3.2] and [10, §4.6], and is essentially a denotational analogue of the expansion law. What remains is the base case, where we obtain $\mathcal{D}yn(X) \times \mathcal{D}yn(Y) \to \mathcal{D}yn(X \times Y)$ from the maps

$$\delta^n(X) \times \delta^m(Y) \longrightarrow \delta^{n+m}(X) \times \delta^{n+m}(Y) \longrightarrow \delta^{n+m}(X \times Y)$$

and the definition of $\mathcal{D}yn(-)$ as a colimit of the $\delta^n$.

For the case of a single $X$ and multiplication $\mu$, we further compose with $Pi(\mu) : Pi(X \times X) \to Pi(X)$ to get $par_\mu$.

The interpretation of specific $\pi$-calculus processes $P$ and $Q$ does not in fact exercise the base case at all, so the soundness of $[\![P \mid Q]\!] = par([\![P]\!], [\![Q]\!])$ is a consequence of Proposition 2 and the exact match between the expansion law and our definition of $par_{X,Y}$. □

In particular there are unique choices for $\mu$ when $X$ is 0 or 1, giving

$$par_0 : Pi(0) \times Pi(0) \longrightarrow Pi(0) \qquad \text{and} \qquad par_1 : Pi(1) \times Pi(1) \longrightarrow Pi(1).$$

Using $par_0$ instead of syntactic expansion in the interpretation of Section 4.1 then gives a purely compositional presentation of the denotational semantics in $Pi(0)$ for finite $\pi$-calculus processes.

In general, appropriate choice of multiplication $\mu$ gives free-algebra models for implementations of the $\pi$-calculus over a set $X$ of basic processes. For example, $Pi(1)$ models the $\pi$-calculus with an extra process "✓" marking completion; from a programming language viewpoint (see Section 4.5 later) this gives a semantics for terminating threads and thread rendezvous.

### 4.4. Fully-abstract $\pi$-algebras

We have now a compositional semantics of the finite $\pi$-calculus in the initial free $\pi$-algebra $Pi(0)$. Expanding the explicit definitions of $Pi$ and $[\![-]\!]$ reveals that this is precisely the fully-abstract model described by Fiore et al. in [10, Thm. 6.4]. Here we extend their analysis to all free $\pi$-algebras.

**Theorem 5.** *For any object $X \in \mathcal{S}et^{\mathcal{I}}$, the free $\pi$-algebra $Pi(X)$ is fully abstract for (strong, late) bisimulation congruence. For all finite $\pi$-calculus processes $P$, $Q$:*

$$P \approx Q \quad \Longleftrightarrow \quad Pi(X) \models P = Q$$

*and hence also:*

$$P \approx Q \quad \Longleftrightarrow \quad \mathcal{S}et^{\mathcal{I}} \models P = Q.$$

**Proof.** The forward direction is Proposition 2, and the reverse direction for $Pi(0)$ comes from the full abstraction result of [10]. We lift this to general $Pi(X)$ by factoring the interpretation $[\![-]\!]_{Pi(X)}$ as $[\![-]\!]_{Pi(0)}$ followed by the monomorphism $Pi(0) \rightarrowtail Pi(X)$. □

Because we reuse an existing full-abstraction result, we have not needed here to inspect the transition behaviour of processes. There is, however, a close correspondence between the possible transitions $P \xrightarrow{\alpha} P'$ of a process $P$, as described in Fig. 2, and the interpretation of $P$ in any free $\pi$-algebra.

Briefly, the correspondence uses the distinct-name interpretation $(\!|s \vdash P|\!)_{Pi(X)} : N^{\otimes|s|} \to Pi(X)$ mentioned after Theorem 2. The source object $N^{\otimes|s|}$ is a representable, so by Yoneda this arrow is equivalent to an element of the set $Pi(X)(s)$; which from (1) is itself isomorphic to $\mathcal{P}_{fin}(s \times s \times Pi(X)(s) \dots)$; and it turns out that the set represented here by $(\!|s \vdash P|\!)$ is exactly the set of transitions out of process $P$. Thus our translation both preserves and reflects the transition semantics (see also [45, Thm. 1]).

### 4.5. Monads and effects for $\pi$

The operations and equations in the theory of $\pi$ fit very well with a process-calculus view of concurrency. However, the monad $T_\pi$ of (1) is also a "computational" monad in the style of Moggi, and gives a programming language semantics of mobile communicating systems. The operations of Section 3 then induce corresponding generic

effects:

$$\begin{array}{rlrl} \mathit{choice} : & A^2 \longrightarrow A & \mathsf{arb} : & 1 \longrightarrow \mathsf{T2} \\ \mathit{nil} : & 1 \longrightarrow A & \mathsf{deadlock} : & 1 \longrightarrow \mathsf{T0} \\ \mathit{out} : & A \longrightarrow A^{N\times N} & \mathsf{send} : & \mathsf{N}\times\mathsf{N} \longrightarrow \mathsf{T1} \\ \mathit{in} : & A^N \longrightarrow A^N & \mathsf{receive} : & \mathsf{N} \longrightarrow \mathsf{TN} \\ \mathit{tau} : & A \longrightarrow A & \mathsf{skip} : & 1 \longrightarrow \mathsf{T1} \\ \mathit{new} : & \delta A \longrightarrow A & \mathsf{fresh} : & 1 \longrightarrow \mathsf{TN}. \end{array}$$

For example, $\mathsf{receive}(\mathsf{c})$ fetches a value from channel $\mathsf{c}$, and $\mathsf{fresh}()$ returns a newly allocated channel. In a suitable computational metalanguage these give a semantics for programming languages that combine higher-order functions with communicating concurrency. Alternatively, they can be used just as they stand in a language like Haskell that explicitly handles computational monads: $\mathsf{do}\{\mathsf{x} \leftarrow \mathsf{receive}(\mathsf{c}); \mathsf{send}(\mathsf{c}', \mathsf{x})\}$.

## 5. Algebras for variations of $\pi$

In this section we look at how small changes to the operations and equations forming the theory of $\pi$ can capture some different versions of the $\pi$-calculus proposed in the literature. We have already seen, at the end of Section 4.1, how changes in the interpretation function $[\![s \vdash P]\!]_A$ can replace bisimulation congruence with other equivalences; we now vary the theory itself to obtain the following:

- A theory $\pi\mathrm{I}$ which replaces the component theory for I/O with one for internal mobility, using a symmetric pair of operations *bin* and *bout*.
- A theory $e\pi$ for the early $\pi$-calculus, which replaces the sum of the component theories for I/O and choice with their commuting combination.

In each case we present the modified theory and discuss the corresponding free-algebra monad.

### 5.1. Internal mobility $\pi\mathrm{I}$

Sangiorgi has identified a distinction between *internal* and *external* mobility in the name-passing operations of the $\pi$-calculus [39]: internal mobility is where a fresh private name is passed from one process to another, while external mobility is the communication of any name that is already known. These correspond directly to the syntactic prefix constructions of bound output $\bar{x}(y).P$ and free output $\bar{x}z.P$ respectively. As noted earlier in Section 3.4, in the full $\pi$-calculus bound output $\bar{x}(y).P$ is a derived operation, equivalent to $\nu y(\bar{x}y.P)$. However, if we make this bound output prefix a primitive, and remove free output, we obtain a subcalculus $\pi\mathrm{I}$ with purely internal mobility.

Perhaps surprisingly, $\pi\mathrm{I}$ retains the considerable expressive power of full $\pi$, and indeed Sangiorgi suggests that: "internal mobility is responsible for much of the expressiveness of the $\pi$-calculus, whereas external mobility is responsible for many of the semantic complications". In particular, there are $\pi\mathrm{I}$ encodings for datatypes, the lambda-calculus, concurrent objects, and agent-passing calculi.

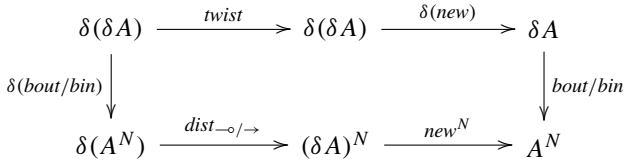
Turning to algebraic models, it is clear that every $\pi$-algebra is already a model for $\pi\mathrm{I}$, using the encoding of bound output from Section 3.4. We can obtain a strictly larger class of models, though, by writing a specific theory of $\pi\mathrm{I}$. This replaces the communication operations of Section 3.2 with the following:

| | |
|---|---|
| $\mathit{bout} : \delta A \longrightarrow A^N$<br>$\mathit{bin} : \delta A \longrightarrow A^N$<br>$\mathit{tau} : A \longrightarrow A$ | (No required equations) |

We can motivate these operations informally:

| | |
|---|---|
| $\mathit{bout}_x(y.p)$ | create a fresh name, transmit it on $x$, then bind to $y$ in process $p$; |
| $\mathit{bin}_x(y.p)$ | receive a name on $x$, certain to be fresh, and bind it to $y$ in process $p$. |

**Commuting component theories**




**Interaction between component theories**

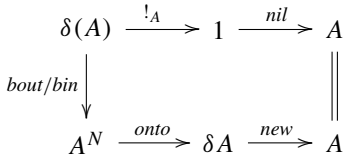



Fig. 6. Commuting diagrams for the theory of internal mobility in $\pi$I. The notation *bout*/*bin* means that each diagram stands for two: one for *bout*, one for *bin*.

Notice that not only is output replaced by a bound version, but so is input, because with internal mobility all names received are sure to be new. This freshness of transmitted names, which is the essence of internal mobility, is captured by the monoidal argument arity on *bout* and *bin*.

We now need equations to combine this theory of internal mobility with the other component theories. These follow closely those for full $\pi$ in Section 3.5: for nondeterministic choice we need only a sum of theories, which requires no additional equations; while dynamic name creation introduces some commutation:

**Commuting component theories**

$$new(x.choice(p,q)) = choice(new(x.p), new(x.q))$$

$$\begin{aligned} new(z.bout_x(y.p)) &= bout_x(y.new(z.p)) && z \neq x \\ new(z.bin_x(y.p)) &= bin_x(y.new(z.p)) && z \neq x \\ new(z.tau(p)) &= tau(new(z.p)) \end{aligned}$$

The first and last equations are as before; commuting diagrams for the middle two are in Fig. 6. Finally, the equations for interaction again involve all three component theories at once.

**Interaction between component theories**

$$\begin{aligned} new(x.bout_x(y.p)) &= nil \\ new(x.bin_x(y.p)) &= nil \end{aligned}$$

Fig. 6 also presents these two as commuting diagrams.

The symmetry of input and output in $\pi$I is readily evident in these equations and their commuting diagrams, which take precisely the same form for *bout* and *bin*.

The development of algebraic models for the theory of $\pi$I then follows much as in Section 4. A $\pi$I-*algebra* in $\mathcal{S}et^{\mathcal{I}}$ is an object $A$ together with maps (*choice*, *nil*, *bout*, *bin*, *tau*, *new*) satisfying the equations for nondeterministic choice (Section 3.1), name creation (Section 3.3), internal mobility (none), and their combination (above). These algebras form a category $\mathcal{PI}_{\mathrm{I}}(\mathcal{S}et^{\mathcal{I}})$, with morphisms the maps $f : A \to B$ that commute with all operations. The forgetful functor $U : \mathcal{PI}_{\mathrm{I}}(\mathcal{S}et^{\mathcal{I}}) \to \mathcal{S}et^{\mathcal{I}}$ takes a $\pi$I-algebra to its carrier object.

Given any $\pi$I-algebra $A \in \mathcal{PI}_{\mathrm{I}}(\mathcal{S}et^{\mathcal{I}})$ we can give a denotational semantics of the finite $\pi$I-calculus, with map

$$[\![s \vdash P]\!]_A : N^{|s|} \longrightarrow A.$$

for any $\pi$I process $P$ with free names in set $s$. The interpretation is as in Fig. 5, but with revised clauses for output and input prefixes:

$$[\![s \vdash \bar{x}(y).P]\!] = bout_x(y.[\![s, y \vdash P]\!]) \qquad [\![s \vdash x(y).P]\!] = bin_x(y.[\![s, y \vdash P]\!]).$$

In diagram form:

$$\frac{[\![s, y \vdash P]\!] : N^{|s|} \times N \longrightarrow A}{\begin{array}{l} [\![s \vdash \bar{x}(y).P]\!] : N^{|s|} \xrightarrow{\langle \ulcorner [\![s,y\vdash P]\!] \circ into \urcorner, proj_x \rangle} \delta A \times N \xrightarrow{bout \times N} A^N \times N \xrightarrow{ev} A \\ [\![s \vdash x(y).P]\!] : N^{|s|} \xrightarrow{\langle \ulcorner [\![s,y\vdash P]\!] \circ into \urcorner, proj_x \rangle} \delta A \times N \xrightarrow{bin \times N} A^N \times N \xrightarrow{ev} A. \end{array}}$$

The interpretation of parallel composition again needs an expansion law for $P \mid Q$; in fact the constraint of internal mobility makes this rather simpler than the standard one [39, §3.3].

As in Proposition 2, we may use an existing axiomatization [39, §3.3] to show that the interpretation respects bisimilarity (which in $\pi$I is a congruence, so there is no need for a separate notion of bisimulation congruence).

The monad for free $\pi$I-algebras is similar to that for $\pi$, with simplifications due to internal mobility:

$$T_{\pi \mathrm{I}}(-) = \mu Y.(\mathcal{P}_{fin}(N \times \delta Y + N \times \delta Y + Y + \mathcal{D}yn(-))). \tag{2}$$

Comparing with (1), the component $N \times N \times Y$ for free output is gone completely, and general input $N \times Y^N$ is replaced by input of fresh names only $N \times \delta Y$. Notice again the complete symmetry between input and output.

As before, given any object $X \in \mathcal{S}et^{\mathcal{I}}$ we can equip $T_{\pi \mathrm{I}}(X)$ with operations to make it the carrier for a free $\pi$I-algebra $Pi_{\mathrm{I}}(X)$. The explicit form (2) for $T_{\pi \mathrm{I}}$ is then enough to show that we have a monadic adjunction $Pi_{\mathrm{I}} \dashv U$ with all free algebras $Pi_{\mathrm{I}}(X)$ fully abstract models for bisimilarity in $\pi$I.

### 5.2. Early semantics of $\pi$

We have so far considered a *late* semantics for the $\pi$-calculus, where processes first commit to communication on a particular channel, and only then pass across a value. In the alternative *early* semantics commitment may also depend on the value passed, leading in particular to a different notion of bisimilarity. Thus, a symmetric relation $\mathcal{S}$ between processes is an *early bisimulation* if for every $(P, Q) \in \mathcal{S}$ the following hold.

- For $\alpha = \tau, \bar{x}y, \bar{x}(y)$, if $P \xrightarrow{\alpha} P'$ then there is $Q'$ such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{S}$.
- If $P \xrightarrow{x(y)} P'$ then for any name $z$ there is a $Q'$ such that $Q \xrightarrow{x(y)} Q'$ and $(P'[z/y], Q'[z/y]) \in \mathcal{S}$.

Compare this to the definition of late bisimulation in Section 2.2. The difference is the order of quantification in the clause for input: in early bisimulation the choice of process $Q'$ may depend on the input value $z$. From this definition follow early bisimilarity $P \sim_E Q$ and early bisimulation congruence $P \approx_E Q$.

Early bisimulation congruence is strictly coarser than the late form: it identifies all late congruent processes, and some others. The situation is therefore the reverse of that for $\pi$I: any algebraic model of early $\pi$ is also a model for late $\pi$, but not a fully abstract one. Here we examine two possible algebraic treatments for early $\pi$: the first directly changes the operations and the interpretation function to early versions; while the second is rather cleaner in just adding one equation to the theory.

The early semantics of $\pi$ can be presented in a transition system with an additional transition $xz$, denoting the input of name $z$ along channel $x$. This uses variants of the input and communication rules:

$$\text{E-IN} \quad x(y).P \xrightarrow{xz} P[z/y] \qquad\qquad \text{E-COM} \quad \frac{P \xrightarrow{xz} P' \quad Q \xrightarrow{\bar{x}z} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}.$$

We can match this early semantics with two new operations:

$$ein : A \longrightarrow A^{N \times N} \qquad \text{and} \qquad choice_N : A^N \longrightarrow A.$$

Here $ein_{x,z}(p)$ represents an early input prefix $xz.P \stackrel{def}{=} x(y).[y = z]P$, which can only make the early transition $xz$. The $N$-fold choice operation strictly extends standard *choice* by its varying arity.

This semantics demands a more elaborate interpretation function for the input prefix, similar to the use of the expansion law for parallel composition:

$$[\![s \vdash x(y).P]\!] = choice_N(ein_{x,z}([\![s \vdash P[z/y]]\!])_z).$$

Here the $N$-fold choice explicitly presents each possible early input transition of the process $x(y).P$.

Such a semantics for early transitions follows that in [5, §5.1], and could give a suitable category of algebras. More interesting than changing the translation, though, is the possibility to leave the operations for $\pi$ untouched and instead adjust only the equations.

Parrow axiomatizes early bisimulation congruence with the following extra equation [29, §9.1]:

$$\text{EARLY} \quad x(y).P + x(y).Q = x(y).P + x(y).Q + x(y).(y = z)\ ?\ P : Q.$$

We could replicate this exactly, but in our setting of component theories there is a more appealing alternative. We take the theories of choice and I/O, and where Section 3 took their sum we instead use their commuting combination. To the equations of Fig. 4 we therefore add the following:

**Commuting component theories**

$$choice(in_x(p_y), in_x(q_y)) = in_x(choice(p, q)_y)$$

$$\begin{array}{ccccc} A^N \times A^N & \xrightarrow{\cong} & (A \times A)^N & \xrightarrow{choice^N} & A^N \\ {\scriptstyle in\times in}\downarrow & & & & \downarrow{\scriptstyle in} \\ A^N \times A^N & \xrightarrow{\cong} & (A \times A)^N & \xrightarrow{choice^N} & A^N \end{array}$$

Informally, we might write this as $x(y).P + x(y).Q = x((y)P + (y)Q)$, although this is not easily expressed in $\pi$-calculus syntax. The intuition is that receiving on a channel $x$ should not force premature choice between alternatives.

This commuting rule is in general stronger than the equation (EARLY), but they coincide on all elements interpreting $\pi$-calculus terms.

From this we proceed as usual: an *$e\pi$-algebra* is an object of $\mathcal{S}et^{\mathcal{I}}$ together with maps satisfying the equation above plus those of Fig. 4; these form a category $\mathcal{PI}_{\mathrm{E}}(\mathcal{S}et^{\mathcal{I}})$. In fact every $e\pi$-algebra is a $\pi$-algebra, $\mathcal{PI}_{\mathrm{E}}(\mathcal{S}et^{\mathcal{I}})$ is a subcategory of $\mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$, and the interpretation of a process $P$ in an $e\pi$-algebra $A$ is exactly the same as that in $A$ considered as a $\pi$-algebra. Note that this includes using the seemingly late operation $in : A^N \to A^N$ to interpret input prefix $x(y).P$.

The free-algebra monad for early $\pi$ is related to that for late $\pi$, but to show this we first rewrite (1) using the isomorphism $\mathcal{P}_{fin}(A + B) \cong \mathcal{P}_{fin}(A) \times \mathcal{P}_{fin}(B)$ to obtain the following:

$$\begin{array}{lll} T_\pi(-) = \mu Y.(\mathcal{P}_{fin}(N \times N \times Y) \times \mathcal{P}_{fin}(N \times \delta Y) & \text{Output} & (3) \\ \quad \times \mathcal{P}_{fin}(N \times Y^N) & \text{Late input} & \\ \quad \times \mathcal{P}_{fin}(Y) \times \mathcal{P}_{fin}(\mathcal{D}yn(-))) & \text{Silent and base case.} & \end{array}$$

At first sight it seems as though an early semantics could be found by replacing the input clause here with a copy of the output clause, using $\mathcal{P}_{fin}(N \times N \times Y)$ for sets of early input actions $xy.P$. However, this does not behave correctly under injective renaming: intuitively, it will not introduce the necessary extra input actions as the set of available names increases.

The correct monad uses a type constructor described by Fiore and Turi [9, §2.2] in their full-abstract model for early $\pi$.

$$\begin{array}{lll} T_{e\pi}(-) = \mu Y.(\mathcal{P}_{fin}(N \times N \times Y) \times \mathcal{P}_{fin}(N \times \delta Y) & \text{Output} & (4) \\ \quad \times N \rightrightarrows (\mathcal{P}_{fin}(Y)^N) & \text{Early input} & \\ \quad \times \mathcal{P}_{fin}(Y) \times \mathcal{P}_{fin}(\mathcal{D}yn(-))) & \text{Silent and base case.} & \end{array}$$

The key term here for early input uses a partial exponential $\rightharpoonup$ defined on $\mathcal{S}et^{\mathcal{I}}$ for any object $A$ and pointed object $B_\perp$ by

$$(A \rightharpoonup B_\perp)(s) = B_\perp(s)^{A(s)} \qquad s \in \mathcal{I}, \quad f : s \to s'$$
$$(A \rightharpoonup B_\perp)(f) : u \longmapsto B_\perp(f) \circ u \circ (A(f))^{-1} \qquad u : A(s) \rightharpoonup B(s).$$

Thus $A \rightharpoonup B_\perp$ is a pointwise exponential, with action on morphisms $f$ determined using the partial inverse $(A(f))^{-1}$. In the particular case here of $N \rightharpoonup (\mathcal{P}_{fin}(Y)^N)$, the object $\mathcal{P}_{fin}(Y)^N$ is pointed with distinguished element the constantly empty set.

An informal interpretation of $N \rightharpoonup (\mathcal{P}_{fin}(Y)^N)$ is that for any currently available channel name $(N \rightharpoonup -)$ it gives a map from each possible input value to a finite set of processes $(\mathcal{P}_{fin}(Y)^N)$. Crucially, the second (non-pointwise) exponential correctly interprets the action of receiving names created at some later stage.

We can draw the similarity between late and early monads even closer by noting that the input clause of $T_\pi$ in (3) is isomorphic to $N \rightharpoonup (\mathcal{P}_{fin}(Y^N))$: the only difference is in the placement of the input value exponential $(-^N)$ being inside or outside the finite powerset. This also gives us a further demonstration that any model for early $\pi$ is also a model for late $\pi$. The evident "map" morphism $\mathcal{P}_{fin}(Y^N) \longrightarrow \mathcal{P}_{fin}(Y)^N$ given by

$$\mathcal{P}_{fin}(Y^N) \ni U \longmapsto \lambda x.\{f x \mid f \in U\} \in \mathcal{P}_{fin}(Y)^N$$

induces a natural transformation $E : T_\pi \to T_{e\pi}$ that for any $T_{e\pi}$-algebra gives a corresponding $T_\pi$-algebra, by precomposition:

$$\begin{array}{ccc} T_\pi(A) & \xrightarrow{E_A} & T_{e\pi}(A) \\ {\scriptstyle E_A;h}\downarrow & & \downarrow{\scriptstyle h} \\ A & = & A \end{array}$$

where $h$ is the structure map for the $T_{e\pi}$-algebra $A$.

## 6. Extensions and further work

So far we have only addressed finite $\pi$-calculus processes, with models in $\mathcal{S}et^{\mathcal{I}}$. There are known fully-abstract models for the full $\pi$-calculus, with replication and recursion, using $\mathcal{C}po^{\mathcal{I}}$; in particular Fiore et al. give a method for lifting full abstraction in $\mathcal{S}et^{\mathcal{I}}$ up to $\mathcal{C}po^{\mathcal{I}}$ [10]. For algebraic theories over domains, Plotkin and Power have already investigated $\mathcal{C}po$-enrichment in work on effects for PCF [34]: in particular, taking the least upper bound of $\omega$-chains is then an algebraic operation of (countable) arity.

Based on these, it seems reasonable to aim at constructing free-algebra domain models for the full $\pi$-calculus. There remain challenges, however: in $\mathcal{C}po^{\mathcal{I}}$ we no longer have convenient element-wise constructions of exponentials and colimits, and there are more "ill-behaved" objects. Section 2.3 mentioned the possibility of working in the Schanuel topos $\mathcal{A}$ rather than $\mathcal{S}et^{\mathcal{I}}$; this is not necessary in the finite case, but for the full $\pi$-calculus it may be sensible to take such a step. Pitts and Shinwell observed similar issues in formulating a notion of "nominal domain" [42,43]; their solution of *FM-Cpos* may well be the right place to work on an algebraic theory of full $\pi$.

Order enrichment also offers the possibility of inequations in theories. For the *choice* operation we can use these to distinguish between upper, lower and convex powerdomains, and conjecture that such theories for $\pi$ could characterize Hennessy's fully-abstract models for must and may-testing [12].

The full-abstraction proofs of Proposition 2 and Theorem 5 make use of existing results on equational axiomatization and full abstraction for the $\pi$-calculus. Given that these results are available, it is only sensible to take advantage of them. However, this dependence is not essential. There is an alternative proof route based on the correspondence between sets of transitions in the operational semantics and the powerset in the free-algebra monad, as outlined at the close of Section 4.4. The procedure follows that in [45]: construct the initial $\pi$-algebra $Pi(0)$; prove that it preserves and reflects transitions; use this to show it also preserves and reflects bisimulation congruence; then use initiality of $Pi(0)$ to deduce that all $\pi$-algebras are sound, and all free-algebras fully abstract. From this one can then

deduce the completeness of the axiomatization given in the theory, as well as the correctness of individual equations, and the expansion rule used in the interpretation.

This "bare-hands" technique would be suitable for $\pi$-calculus variants where the necessary results, such as a complete equational axiomatization, are not already known. Notice though that the key step of giving an explicit free-algebra monad corresponds anyway to the usual problem in an operational approach of identifying suitable normal forms for processes.

Section 5 showed how to modify the component theories of $\pi$ to model internal mobility in $\pi\mathrm{I}$ and early bisimilarity in $\pi$. The same approach might be applied to other $\pi$-calculus equivalences like open and weak bisimilarity, known to be challenging for denotational semantics. Equational axiomatizations are available for both of these [29, §9], and we now need to explore the algebraic theories they generate.

The asynchronous $\pi$-calculus of Boudol [3] and Honda and Tokoro [13] replaces output prefix $\bar{x}\,y.P$ with a simple output atom $\bar{x}\,y$, with the effect that no process can block on an output action. This is a very expressive fragment of $\pi$, though known to be strictly less powerful than the full calculus [27]. Such a simplification naturally suggests a theory based on an operation $aout : 1 \to A^{N\times N}$; however this falls down due to the absence of an expansion law in asynchronous $\pi$. This might seem trivial, but the deeper problem is that all of our models here treat process behaviour as a tree of guarded choices, and these synchronisation trees are incompatible with an asynchronous calculus.

Work of Amadio et al. [2] suggests a possible approach to solve this. They observe that standard bisimilarity is not, in fact, an appropriate notion for the asynchronous $\pi$-calculus, as it treats input actions as observable. To replace this they develop a distinct theory of asynchronous bisimulation, together with an equational axiomatization and appropriate normal forms. This looks promising for our algebraic formulation, but some issues remain: in particular as the normal forms combine sums with parallel composition of output atoms.

As a further variation, Palamidessi et al. [28] distinguish *linear* and *persistent* asynchronous $\pi$-calculus, using replicated inputs and outputs. It seems reasonable to ask whether an algebraic theory of asynchronous $\pi$ could also characterise these different classes of behaviour.

Section 2.3 mentioned that Pitts and others have championed *nominal sets* and Fraenkel–Mostowski set theory as a foundation for reasoning with names [11,30,42]. We noted there that all of our constructions so far lie within the subcategory of pullback-preserving functors in $\mathcal{S}et^{\mathcal{I}}$, which is equivalent to the Schanuel topos $\mathcal{A}$ and gives a model of FM set theory. From this we conjecture that our $\pi$-calculus models are examples of universal algebra within FM set theory – given first an investigation of just what that is.

At the end of Section 4.3 we mentioned the use of $Pi(1)$ to model the $\pi$-calculus with an additional basic process "$\checkmark$" to show successful completion. More generally, other free algebras $Pi(X)$, and indeed the full range of $\pi$-algebras in $\mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$, may be useful to model applications of the $\pi$-calculus with domain-specific terms, equations and processes. There are many such ad-hoc extensions, notably those brought together by Abadi and Fournet under the banner of *applied* $\pi$ [1].

In ongoing work, Plotkin and Power have given a construction for modal logics from algebraic theories [35]. Applying this to the theory of $\pi$ gives a modal logic for the $\pi$-calculus up to bisimulation congruence. This can represent Hennessy–Milner logic, and also has modalities for choice and name creation; though no "spatial" modality for parallel composition.

We can extend our notion of $\pi$-algebra to other categories $\mathcal{C}$, enriched over $\mathcal{S}et^{\mathcal{I}}$. The written form for the theory of $\pi$ remains the same as in Section 3, but its interpretation is a little richer: for example, now $A^N$ stands not for the function space $N \to A$ in $\mathcal{S}et^{\mathcal{I}}$, but the *cotensor* of an object $A \in \mathcal{C}$ by arity $N \in \mathcal{S}et^{\mathcal{I}}$. The category $\mathcal{C}$ must have such cotensors (both cartesian and monoidal) for every finitely presentable object of $\mathcal{S}et^{\mathcal{I}}$ [36]. However, we do not yet have conditions for the existence of free algebras, or for full abstraction, in general $\mathcal{C}$. This would require further investigation of the properties of algebras enriched over a doubly closed structure, as in $\mathcal{S}et^{\mathcal{I}}$.

An alternative path, following a suggestion of Fiore, is to give a theory of name testing that exhibits $\mathcal{S}et^{\mathcal{I}}$ as monadic over $\mathcal{S}et^{\mathcal{F}}$, where $\mathcal{F}$ is the category of finite name sets and all maps. We have a candidate theory, and conjecture that in combination with our existing theory of $\pi$, this would allow us to generate algebraic models of $\pi$ in $\mathcal{S}et^{\mathcal{F}}$ using only cartesian closed structure. This is particularly appealing in that it would move some of the implicit name handling infrastructure – monoidal $\otimes$, $\multimap$ and shift $\delta$ – into the explicit theory, making it available for further inspection and modification.

## Acknowledgements

Thanks to Gordon Plotkin and John Power for discussions motivating this work, and patient explanation of the details of enriched Lawvere theories; also to the anonymous referees for their comments, in particular leading to the work in Section 5.

This research was supported by EPSRC Advanced Research Fellowship GR/R76950/01 *Mathematical Models for Concurrent and Mobile Computation*, and in part by the *Mobius* project IST-015905 under the 6th Framework Information Society Technologies programme of the European Commission, who require the following disclaimer: this article reflects only the author's views and the European Community is not liable for any use that may be made of the information contained therein.

## References

[1] Martín Abadi, Cédric Fournet, Mobile values, new names, and secure communication, in: Conference Record of POPL '01: 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM Press, 2001, pp. 104–115.
[2] Roberto M. Amadio, Ilaria Castellani, Davide Sangiorgi, On bisimulations for the asynchronous $\pi$-calculus, Theoretical Computer Science 195 (2) (1998).
[3] Gérard Boudol, Asynchrony and the $\pi$-calculus, Rapport de recherche 1702, INRIA, Sophia Antipolis, 1992.
[4] Gian Luca Cattani, Peter Sewell, Models for name-passing processes: Interleaving and causal, Information and Computation 190 (2) (2004) 136–178.
[5] Gian Luca Cattani, Ian Stark, Glynn Winskel, Presheaf models for the $\pi$-calculus, in: Category Theory and Computer Science: Proceedings of the 7th International Conference, CTCS '97, in: Lecture Notes in Computer Science, vol. 1290, Springer-Verlag, 1997, pp. 106–126.
[6] Brian J. Day, On closed categories of functors, in: Reports of the Midwest Category Seminar IV, in: Lecture Notes in Mathematics, vol. 137, Springer-Verlag, 1970, pp. 1–38.
[7] Marcelo Fiore, Gordon Plotkin, Daniele Turi, Abstract syntax and variable binding, in: Proceedings of the Fourteenth Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, 1999, pp. 193–202.
[8] Marcelo Fiore, Sam Staton, Comparing operational models of name-passing process calculi, in: CMCS 2004: Proceedings of the 7th International Workshop on Coalgebraic Methods in Computer Science, in: Electronic Notes in Theoretical Computer Science, vol. 106, Elsevier, 2004, pp. 91–104.
[9] Marcelo Fiore, Daniele Turi, Semantics of name and value passing, in: Proceedings of the Sixteenth Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, 2001, pp. 93–104.
[10] Marcelo P. Fiore, Eugenio Moggi, Davide Sangiorgi, A fully-abstract model for the $\pi$-calculus, Information and Computation 179 (1) (2002) 76–117.
[11] Murdoch J. Gabbay, Andrew M. Pitts, A new approach to abstract syntax with variable binding, Formal Aspects of Computing 13 (3–5) (2001) 341–363.
[12] Matthew Hennessy, A fully abstract denotational semantics for the $\pi$-calculus, Theoretical Computer Science 278 (1–2) (2002) 53–89.
[13] Kohei Honda, Mario Tokoro, An object calculus for asynchronous communication, in: ECOOP '91: Proceedings of the European Conference on Object-Oriented Programming, in: Lecture Notes in Computer Science, vol. 512, Springer-Verlag, 1991, pp. 133–147.
[14] Martin Hyland, Gordon Plotkin, John Power, Combining computational effects: Commutativity and sum, in: Proceedings of the 2nd IFIP International Conference on Theoretical Computer Science, TCS 2002, Kluwer, 2002, pp. 474–484.
[15] Martin Hyland, Gordon Plotkin, John Power, Combining effects: Sum and tensor, Theoretical Computer Science 357 (1–3) (2006) 70–99. Journal version of [14].
[16] Mark P. Jones, Luc Duponcheel, Composing monads, Research Report YALEU/DCS/RR-1004, Yale University Department of Computer Science, 1993.
[17] G.M. Kelly, A. John Power, Adjunctions whose counits are coequalizers, and presentations of finitary enriched monads, Journal of Pure and Applied Algebra 89 (1993) 163–179.
[18] Sheng Liang, Paul Hudak, Mark P. Jones, Monad transformers and modular interpreters, in: Conference Record of POPL '95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM Press, 1995, pp. 333–343.
[19] Saunders Mac Lane, Ieke Moerdijk, Sheaves in Geometry and Logic: A First Introduction to Topos Theory, Springer-Verlag, 1992.
[20] Robin Milner, Communicating and Mobile Systems: The Pi-Calculus, Cambridge University Press, 1999.
[21] Eugenio Moggi, An abstract view of programming languages, Technical Report ECS-LFCS-90-113, Laboratory for Foundations of Computer Science, University of Edinburgh, 1990.
[22] Eugenio Moggi, Notions of computation and monads, Information and Computation 93 (1) (1991) 55–92.
[23] Jeff Newburn, All about monads, v1.1.0. http://www.nomaware.com/monads.
[24] Peter W. O'Hearn, R.D. Tennent, Parametricity and local variables, Journal of the ACM 42 (3) (1995) 658–709. Reprinted in [25].
[25] Peter W. O'Hearn, Robert D. Tennent (Eds.), Algol-Like Languages, Birkhauser, 1996.
[26] Frank J. Oles, Functor categories and store shapes. Chapter 11 of [25].
[27] Catuscia Palamidessi, Comparing the expressive power of the synchronous and the asynchronous $\pi$-calculus, Mathematical Structures in Computer Science 13 (5) (2003) 685–719.

[28] Catuscia Palamidessi, Vijay Saraswat, Frank D. Valencia, Björn Victor, On the expressiveness of linearity vs persistence in the asynchronous $\pi$-calculus, in: Proceedings of the Twenty-First Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, 2006.
[29] Joachim Parrow, An introduction to the $\pi$-calculus, in: Handbook of Process Algebra, Elsevier, 2001, pp. 479–543.
[30] Andrew M. Pitts, Nominal logic, a first order theory of names and binding, Information and Computation 186 (2003) 165–193. Errata, September 2004.
[31] Andrew M. Pitts, Ian Stark, Observable properties of higher order functions that dynamically create local names, or: What's *new*? in: Mathematical Foundations of Computer Science: Proceedings of the 18th International Symposium, MFCS '93, in: Lecture Notes in Computer Science, vol. 711, Springer-Verlag, 1993, pp. 122–141.
[32] Gordon Plotkin, A. John Power, Computational effects and operations: An overview, Laboratory for Foundations of Computer Science, University of Edinburgh. Edinburgh Research Archive: http://hdl.handle.net/1842/198, 2002.
[33] Gordon Plotkin, John Power, Notions of computation determine monads, in: Foundations of Software Science and Computation Structures: Proceedings of the 5th International Conference, FoSSaCS 2002, in: Lecture Notes in Computer Science, vol. 2303, Springer-Verlag, 2002, pp. 342–356. Erratum, August 2002.
[34] Gordon Plotkin, John Power, Algebraic operations and generic effects, Applied Categorical Structures 11 (1) (2003) 69–94.
[35] Gordon Plotkin, John Power, Logic for computational effects: Work in progress. Available online at: http://homepages.inf.ed.ac.uk/gdp/publications/, December 2003.
[36] A. John Power, Enriched Lawvere theories, Theory and Applications of Categories 6 (7) (1999) 83–93.
[37] John C. Reynolds, The essence of Algol, in: Proceedings of the 1981 International Symposium on Algorithmic Languages, North-Holland, 1981, pp. 345–372. Reprinted in [25].
[38] Edmund Robinson, Variations on algebra: Monadicity and generalisations of equational theories, Formal Aspects of Computing 13 (3–5) (2002) 308–326.
[39] Davide Sangiorgi, $\pi$-calculus, internal mobility and agent-passing calculi, Theoretical Computer Science 167 (1–2) (1996) 235–274.
[40] Davide Sangiorgi, David Walker, The $\pi$-Calculus: A Theory of Mobile Processes, Cambridge University Press, 2001.
[41] Ulrich Schöpp, Ian Stark, A dependent type theory with names and binding, in: Computer Science Logic: Proceedings of the 18th International Workshop, CSL 2004, in: Lecture Notes in Computer Science, vol. 3210, Springer-Verlag, 2004, pp. 235–249.
[42] Mark R. Shinwell, Andrew M. Pitts, On a monadic semantics for freshness, Theoretical Computer Science 342 (1) (2005) 28–55.
[43] Mark R. Shinwell, Andrew M. Pitts, Murdoch J. Gabbay, FreshML: Programming with binders made simple, in: ICFP 2003: Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming, ACM Press, 2003, pp. 263–274. Erratum, May 2004.
[44] Ian Stark, Categorical models for local names, LISP and Symbolic Computation 9 (1) (1996) 77–107.
[45] Ian Stark, A fully abstract domain model for the $\pi$-calculus, in: Proceedings of the Eleventh Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, 1996, pp. 36–42.
[46] Ian Stark, Free-algebra models for the $\pi$-calculus, in: Foundations of Software Science and Computation Structures: Proceedings of the 8th International Conference, FOSSACS 2005, in: Lecture Notes in Computer Science, vol. 3441, Springer-Verlag, 2006, pp. 155–169.
[47] Philip Wadler, David King, Combining monads, in: Proceedings of the 1992 Glasgow Workshop on Functional Programming, Springer-Verlag, 1993, pp. 134–143.